Proceedings of the IASTED International Conference
Modeling and Simulation (MS '99)
May 5-8, 1999, Philadelphia, Pennsylvania - USA

A NEURO-FUZZY MODEL FOR NONLINEAR PLANTS IDENTIFICATION

Ieroham Baruch*-**, Elena Gortcheva**,
Federico Thomas*** and Ruben Garrido**

**Institute of Information Technologies,*
Bulgarian Academy of Sciences, Akad. G. Bonchev str.,
BAS bl. 29 A, Sofia 1113, BULGARIA;
Fax: (+359-2)72-04-97; E-mail: ips@bgearn.acad.bg
***CINVESTAV-IPN, Ave. IPN No 2508,*
A.P. 14-470 Mexico D.F., MEXICO
Fax: (+52-5)747-70-89; E-mail baruch@ctrl.cinvestav.mx
****IRI-UPC, CSIC, Gran Capitan 2-4, edif. Nexus, pl.2,*
E-08034 Barcelona, SPAIN
Fax: +34-934015750; E-mail: thomas@iri.upc.es

Abstract. A improved parallel Recurrent Neural Network (RNN) model and an improved dynamic Back-propagation (BP) method of its learning, are proposed. The RNN model is given as a two layer Jordan canonical architecture for both continuous and discrete-time cases. The output layer is of Feedforward type. The hidden layer is a recurrent one with self-feedbacks and full forward connections with the inputs. A linearisation of this RNN model is performed and the stability, observability and controllability conditions, are studied. To preserve the RNN stability, sigmoid activation functions are introduced in RNN feedback loops. The paper suggests to improve RNN realisation using saturation function instead of a sigmoid one. A new improved RNN learning algorithm of dynamic BP-type containing momentum term, is proposed. For a complex non-linear plants identification, a fuzzy-rule-based system and a neuro-fuzzy model, are proposed. The proposed neuro-fuzzy model is applied for identification of a mechanical system with friction.

Keywords: recurrent neural network, Jordan canonical form, dynamic backpropagation learning, neuro-fuzzy model, mechanical system identification.

1. INTRODUCTION

The Neural Network (**NN**) modelling and application to system identification, prediction and control was discussed for many authors [1], [2], [3], [4], [5], [6], [7]. Mainly, two types of **NN** models are used: Feedforward (**FFNN**) and Recurrent (**RNN**). The main problem here is the use of different **NN** mathematical descriptions according to the structure of the plant model. For example, Narendra and Parthasarathy, [1], [2], applied **NN** for system identification and control using **FFNN** for various non-linear plants. They considered four plant models with a given structure and supposed that the order of the plant dynamics is known. Yip and Pao, [3] solved control and prediction problems by means of a flat-type **FFNN**. Connor, [4] applied Nonlinear Autoregressive Moving Average (**NARMA**) **RNN** model for robust time series prediction. Pham, [6] applied Jordan **RNN** for robot control. Sastry, [7] introduced two types of neurones - Network Neurones and Memory Neurones to solve identification problems, considering that the plant model is also autoregressive one. Tsoi, [7] summarised different **RNN** models to solve identification and control problems. Baruch et all, [8], in their previous paper, applied the state-space approach to describe **RNN** in an universal way, defining a Jordan canonical two- or three-layer **RNN** model, named Recurrent Trainable Neural Network (**RTNN**) and studied its stability by means of the first stability law of Liapunov. Further, they propose a Fuzzy-Neural (**FN**) approach and Takagi-Sugeno model to resolve more complex identification tasks, [9]. The aim of this paper is to perform a deep topological analysis of the described models and to propose an improved **RTNN** model, introducing some feedback restrictions to preserve **RTNN** stability and to confirm its identification abilities. Simultaneously with the improvement of **RTNN** topology, some advanced researches has been done on the methods of its learning, leading to the design of a new **RTNN** learning method and to its new neuro-fuzzy implementation [8], [9]. The proposed **RTNN** model for system identification is studied by means of various non-linear discrete-time dynamic objects. Three types of object models are suggested: models, non-linear on their output, state and input. A new **MATLAB** - based functions, designed for this aim, is used. Simulation examples of nonlinear object and of mechanical object with friction,

[10], identified by two **RTNN** models, coordinated by a rule-based fuzzy system (**RBFS**), is given.

2. DESCRIPTION OF THE RTNN MODEL

Drawbacks of all described in the literature **NN** models are:

1. There exists a great variety of **NN** models and a universality is missing, [1]; [2], [3], [4], [5], [6], [7];
2. All **NN** models are sequential in nature as implemented for systems identification. (The **FFNN** model uses one or two tap-delays in the input, [1], [2] and **RNN** models usually are based on the autoregressive model, [4], [7] which is one-layer sequential one);
3. Some of the applied **RNN** models are not trainable, [7] - others are not trainable in the feedback part, [6]. Most of them are dedicated to a **SISO** and not to a **MIMO** applications, [7];
4. In more of the cases, the stability of the **RNN** is not considered, especially during the learning.
5. In the case of **FFNN** application for systems identification, the plant is given in one of the four described in [1] plant models. The linear part of the plant model, especially the system order, has to be known and the **FFNN** approximates only the non-linear part of the plant model, [1].

The paper proposes an improved universal parallel **RNN** canonical architecture (named Recurrent Trainable Neural Network - **RTNN**), suited for state-space control systems design. It is easy to extend **RTNN** architecture in a multimodel sense using the **FN** approach.

To describe the **RTNN** model, let us consider the continuous **RNN** linearised mathematical model, defined in [7], and written in state-space form as:

$$\dot{\mathbf{x}} = \mathbf{Ax} + \mathbf{Bu} \quad (1)$$

$$\mathbf{y} = \mathbf{Cx} \quad (2)$$

where **x** - is a **n** -state vector; **u** - is an **m**- input vector; **y** - is an **l**- output vector; **A** is a full (**nxn**) constant state matrix, which diagonal elements represent the passive decay rate dynamic of each neurone and the other elements represent the lateral feedback between neurones; **B** is a (**nxm**) input matrix; **C** is a (**lxn**) output constant matrix. This mathematical model is sufficient for dynamic point of view because, if the linearised model is stable - as the sigmoid function **S(x)** is a single decreasing and bounded - than the non-linear model will also be stable (as follows from the first stability law of Liapunov). The discrete-time **RNN** linearised model could be described using the same state-space mathematical model, as it is:

$$\mathbf{X(k+1) = GV(k)+HU(k),\ k=0,1....,N} \quad (3)$$

$$\mathbf{Y(k) = CX(k)} \quad (4)$$

where **X, Y, U** are state, output and input **n, l, m** - vectors respectively, **k** is a discrete - time integer variable; **G, H, C** are constant matrices of compatible dimensions which depends on the period of discretisation **To**, as it is:

$$\mathbf{G(To) = ATo;\ H(To)= B(To)} \quad (5)$$

The **RNN** linearised state-space models (1), (2) and (3), (4), respectively, could be transformed into Jordan Canonical Form (**JCF**), using similar canonical transformation:

$$\dot{\mathbf{v}} = \mathbf{Jv + Du},\quad \mathbf{x = Tv} \quad (6)$$

$$\mathbf{y = Fv} \quad (7)$$

$$\mathbf{V(k+1) = JV(k)+DU(k),\ X(k)=TV(k)} \quad (8)$$

$$\mathbf{Y(k) = FV(k)} \quad (9)$$

where **v, y, u** are respectively **n, l ,m** - vectors, **T** is a (**nxn**) transformation matrix, **J** is a block-diagonal constant state matrix; **D, F** are also constant matrices with compatible dimensions. In the discrete-time case **J, D, F** depends on **To**. The transformation could be done using eigenvalue/eigenvector method for diagonalisation of the state matrix **A**. This similar canonical transformation not affects the dynamic input/output behaviour of the **RNN** model, but it affects only the state vector **x** , optimising the **RNN** topology in some sense. The continuous-time **RNN** model will be stable if the eigenvalues of **A**, which appear in the main diagonal of **J**, have negative real parts. The discrete-time model will be stable if these eigenvalues are inside the unit circle. Than it is easy to analyse the **NN** model controllability, observability and identifiability of both models. Last concept, taken from systems theory, give us the possibility to check if the obtained global **RNN** model could be learned or not. From the block structure of **D** and **F**, corresponding to the block structure of **J**, we can conclude that: if the input matrix **D** has zero blocks - the **RNN** continuous or discrete-time model is uncontrollable and if the transpose of the output matrix **F** has zero blocks - the **RNN** model is unobservable (if one of both occurs the **RNN** model is unidentifiable). The analysis of the **RNN** model controllabality / observability (identifiability / learnability) and stability give us the possibility to obtain full information of the dynamic behaviour and learning capability of the **RNN**. Based on the Jordan **RNN** canonical state-space representation, it is easy to describe the continuous and discrete-time **RTNN** model, as it was defined by Baruch et all.,[8].

Let us form two layers in the continuous and discrete-time versions of the **JCF RNN** representation (6), (7) and (8), (9), introducing two vector valued sigmoid functions and two new variables **w,z,** (**W, Z**) in the hidden and output layers. So, it is possible to describe the two layer **JCF** architecture of the **RTNN,** which. continuous and discrete-time **JCF RTNN** models, are given in the form:

$$\dot{\mathbf{v}} = \mathbf{Jv} + \mathbf{Du},\ \mathbf{w} = \mathbf{S(v)},\ \mathbf{y} = \mathbf{Fw},\ \mathbf{z} = \mathbf{S(y)} \quad (10)$$

$$\mathbf{V(k+1)} = \mathbf{JV(k)} + \mathbf{DU(k)},\ \mathbf{W(k)=S[V(k)]},$$
$$\mathbf{Y(k)} = \mathbf{FW(k)},\quad \mathbf{Z(k)=S[Y(k)]} \quad (11)$$

where: **w**, **z**, (**W**, **Z**) are new variables with **n**, **l**-dimensions, respectively and **S(x)** is a vector valued sigmoid function, given by:

$$\mathbf{S'(x)} = [\mathbf{s}(x_1), \mathbf{s}(x_2),...,\mathbf{s}(x_j)] \quad (12)$$

with elements, equal to:

$$\mathbf{s(inp)} = 1/[1+\exp(-\mathbf{inp})];\ \mathbf{inp}=\sum_i(\mathbf{d}_i x_i+\mathbf{d}_{io}) \quad (13)$$

Here: **inp** is the input of the sigmoid function; $\mathbf{d}_i$, $\mathbf{d}_{io}$ are trainable weights of the **RTNN**; **S'(x)** signifies a vector transpose of **S(x)**.

The main advantages of the proposed two layer **JCF RTNN** architecture, defined by (10) or (11) are:

1. The described Jordan Canonical **RNN** model is an universal recurrent neural model, applicable for systems identification of various types non-linear plants models. It has a minimum number of parameters and it is completely parallel as the Jordan canonical form is parallel compared with the regressive model, which is a sequential one;

2. The described **RTNN** model is a two-layer hybrid **JCF** model, which contains one output **FF** layer and one hidden recurrent layer with completely decomposed dynamics as the **J** matrix is a diagonal one;

3. It is described in state-space form (**SISO** or **MIMO**) and could serve as one-step ahead state predictor/estimator;

4. The **RTNN** model is non-linear in large and linear in small, so the matrices **J**, **D**, **F** obtained as a result of learning, could be used for analytical design of linear state/output control laws. By means of a similar transformation the **JCF** could be transformed into Luenberger's Canonical Form which is easy to use for pole assignment design of control systems . The matrices **J**, **D** could be used for an optimal control systems design with quadratic performance index. The matrices **J**, **D**, **F** also could be used for an optimal **P, PI**, **PID** control systems design . Finally, the matrices **J**, **D**, **F** could be used in an adaptive iterative algorithm for optimal control with quadratic cost criterion ;

5. The **RTNN** could solve the optimal control problem itself by means of **NN** mapping , [2];

6. The obtained **RTNN** model is a robust model, as the learning method applied for weights adjustment is a Backpropagation-like method, which is based on the network sensitivity model, [2]. It also permits to perform node pruning and weight fixing during the learning.

It was assumed that each Jordan block of the **JCF RTNN** has only (**1**x**1**) or (**2**x**2**) dimension. To preserve the **RTNN** stability during the learning, it is necessary to impose some restrictions on the model feedback, introducing a sigmoid vector function there. So, equations. (10) and (11) are changed in the form:

$$\dot{\mathbf{v}} = \mathbf{S(Jv + Du)},\ \mathbf{w} = \mathbf{S(v)},\ \mathbf{y} = \mathbf{Fw},\ \mathbf{z} = \mathbf{S(y)} \quad (14)$$

$$\mathbf{V(k+1)} = \mathbf{S[JV(k) + DU(k)]},\ \mathbf{W(k)=S[V(k)]},$$
$$\mathbf{Y(k)} = \mathbf{FW(k)},\ \mathbf{Z(k)=S[Y(k)]} \quad (15)$$

Another improvement of the **RTNN** architecture is to facilitate its realisation, approximating the sigmoid function **s(inp)** with a saturation:

$$\mathbf{sat(inp)} = \begin{cases} +1, & \mathbf{inp} \geq +1 \\ \mathbf{inp}, & 0 \leq \mathbf{inp} < +1 \\ 0, & \mathbf{inp} < 0 \end{cases} \quad (16)$$

Both improvements of the **RTNN** architecture should be experimented by simulation examples during the learning.

3. RTNN LEARNING

The most common used **BP** updating rule, applied for the two-layer **RTNN** canonical model, is the following:

$$\mathbf{Q_{ij}(k+1)=Q_{ij}(k)} + \eta\ \Delta\mathbf{Q_{ij}(k)} \quad (17)$$

where: $\mathbf{Q_{ij}}$ is the **ij**-th weight element of each weight matrix in the **RTNN** model to be updated; $\Delta\mathbf{Q_{ij}}$ is the weight correction of $\mathbf{Q_{ij}}$; η is the learning rate parameter. Note that the **RTNN** model weight matrices here are denoted by **Q** for sake of generality.

The weight corrections of the updated matrices in the discrete-time **RTNN** model, described by eqn. (11), are performed using the following update equations:

- For the output layer:

$$\Delta\mathbf{F_{ij}(k)} = [\mathbf{T_j(k)}-\mathbf{Z_j(k)}]\ \mathbf{Z_j(k)}\ [\mathbf{1}-\mathbf{Z_j(k)}]\ \mathbf{W_i(k)} \quad (18)$$

where: $\Delta\mathbf{F_{ij}}$ is the weight correction of the **ij**-th elements of the (**l**x**n**) learned matrix **F**; $\mathbf{T_j}$ is a **j**-th element of the target vector; $\mathbf{Z_j}$ is a **j**-th element of the output vector; $\mathbf{W_i}$ is an **i**-th element of the output vector of the hidden layer.

- For the hidden layer:

$$\Delta\mathbf{D_{ij}(k)} = \mathbf{R}\ \mathbf{U_i(k)} \quad (19)$$
$$\Delta\mathbf{J_{ij}(k)} = \mathbf{R}\ \mathbf{V_i(k-1)} \quad (20)$$
$$\mathbf{R} = \mathbf{F_i(k)}\ [\mathbf{T(k)}-\mathbf{Z(k)}]\ \mathbf{W_j(k)}\ [\mathbf{1}-\mathbf{W_j(k)}] \quad (21)$$

where: $\Delta\mathbf{D_{ij}}$ is the weight correction of the **ij**-th elements of the (**m**x**n**) learned matrix **D**; $\mathbf{F_i}$ is a row vector of dimension (**1**x**l**), taken from the transposed matrix **F'**; **[T-Z]** is a (**l**x**1**) output error vector, through which the error is backpropagated to the hidden layer; $\mathbf{U_i}$ is an **i**-th element of the input vector **U**; $\mathbf{V_i}$ is an **i**-th element of the

vector **V**; $\Delta \mathbf{J_{ij}}$ is the weight correction of the **ij**-th elements of the (**n**x**n**) block-diagonal matrix **J** under learning; **R** is an auxiliary variable. Note that the matrix elements of **0** and **1** values will not be updated. The same equation for **RTNN** learning may be applied for the continuous-time case, given by eqn. (10).

An improvement of the **BP** updating algorithm (17) is to introduce a momentum term, proportional to the past (**k-1**)-th weight correction, as it is:

$$\mathbf{Q_{ij}(k+1) = Q_{ij}(k) + \eta\, \Delta Q_{ij}(k) + \alpha\, \Delta Q_{ij}(k-1)} \qquad (22)$$

where: α is a momentum learning rate parameter. This correction is appropriate to perform in the case when significant error-function oscillations occur. A lot of experiments of learning with different rates of learning η and α has been done. These experiments show that the optimal combination of these learning parameters is obtained when the following inequality condition yields:

$$\mathbf{r_{max} < sqrt(\eta^2+\alpha^2) < 1;\ r_{max} = max|\lambda_i|} \qquad (23)$$

where $\mathbf{max|\lambda_i|}$ is the maximum eigenvalue of the identified plant model. The experimentally obtained rule (23) shows that the circle with radius **r**, which depends on the values of η and α ($\mathbf{r=sqrt(\eta^2+\alpha^2)}$), must contain all eigenvalues of the matrix **J** of the discrete-time **RTNN** model (11).

Another improvement of the **RTNN** learning algorithm, successfully applied for the **BP** learning of discrete-time **RTNN**s consider unimportant units pruning and non-useful connections removing. Both methods remove the units or the weights, whose outputs or values tend to zero. Simultaneously with the nodes pruning, a weight fixing could be applied.

Both learning algorithms, performing weights pruning and fixing lead to exclusion of weights or nodes from the process of learning. There are two possibilities: to fix some weights or to fix the whole node. The first is more efficient then the second because in this case we do not need to compute the node error. The improved learning algorithm for **RTNN** was tested with several linear and non-linear dynamic objects. The topology improvements are also carefully studied. The applicability of the improved **RTNN** model for system identification and prediction is illustrated by appropriate example of non-linear dynamic system.

4. A FUZZY-SYSTEM RULE-BASED COORDINATION MODEL

Let us assume that the unknown system **y = f(x)** generates the data **y(k)** and **x(k)** measured at **k, k-1,..,** then the aim is to use this data to construct a deterministic function **y = F(x)** that can serve as a reasonable approximation of **y = f(x)** in which the function **f** is unknown. The variables $\mathbf{x = [x1,...,xp]'} \in \aleph \subset \Re^p$ and $\mathbf{y} \in Y \subset \Re$ are called regressor and regressand, respectively. In Fuzzy System (**FS)** modelling, the function **F** is represented as a collection of if-than fuzzy rules, represented by the statement:

IF antecedent proposition **then** consequent proposition

The Takagi-Sugeno model, cited in [9], is a mixture between a linguistic and mathematical regression models. The rule antecedents describes the fuzzy regions in the input space. The rule consequent is crisp mathematical function of the inputs. The **FS** model have the most general form:

$$R_i: \quad \textbf{If } x \text{ is } A_i \textbf{ then } y_i = f_i(x),\ i=1,2,.. \qquad (24)$$

Having in mind the linearised version of the **RTNN** model (8), (9), this fuzzy model could be rewritten in the form:

$$R_i: \ \textbf{If } x(k) \textbf{ is } J_i \text{ and } u(k) \textbf{ is } D_i$$

$$\textbf{then} \begin{cases} x_i(k+1) = J_i\, x(k) + D_i\, u(k) \\ yi(k) \quad = F_i\, x(k) \end{cases} \qquad (25)$$

So, it is clear that this fuzzy model combines a fuzzy antecedent proposition with a deterministic mathematical equation describing the consequent part of the rule [9]. Now, it is possible to incorporate the **RTNN** model (23) into the **FS** model (25), using the linear part of the **RTNN** model in the consequent proposition part of (25) and the restrictions, expressed by the saturation of (23) to construct the antecedent proposition part of (25). The biases, obtained in the process of **BP** learning of the **RTNN** model could be used to form the membership functions, as they are natural centres of gravity, [9], for each variable. The number of rules could be optimised using the mean-square error (**MSE**) of **RTNNs** learning, which is a natural measure of the precision of the approximation of the non-linear object model (**MSE** is up to 5%). The structure of the entire identification system contains a fuzzyfier a Fuzzy Rule-Based System (**FRBS)**, and a set of **RTNN** models. The system does not need a defuzzyfier, because **RTNN** models are crisp limited linear state-space models. A possible adaptive control system, could contain also a set of controllers incorporated in a **FRBS**, designed on the base of the obtained set of **RTNN** models.

5. SIMULATION RESULTS

For sake of a **RTNN** and a neuro-fuzzy simulation, various m-modules of **MATLAB**-based functions, has been designed. These functions are allocated in the **Simdemos** directory of **MATLAB** and consists of: fifteen main programs, named **dnn1.m** to **dnn15.m**; eight functions, simulating different plant models, named

object1.m to **object8.m**; three functions, simulating different learning and test input signals with one epoch length, named **signal1.m** to **signal3.m**; two functions, computing **RTNN** output (forward step of the **BP** learning algorithm), named **frwd1.m** and **fwrd2.m**; two functions, computing **RTNN** weight updates (backward step of the **BP** learning algorithm), named **back1.m** and **back2.m** and three functions, simulating different activation functions (sigmoid, hyperbolic tangent and saturation), used in forward step of **BP** learning and named **actf1.m** to **actf3.m**. Two representative examples has been chosen. First example is a discrete-time model of nonlinear object with input and output smooth nonlinearities, identified by one **RTNN** model. The learning signal is chosen as sequence of pulses with random amplitude and width. The neural model quality is verified comparing reactions of the plant and the **RTNN** to an unknown test signal of one-epoch length. These signal has the form, [5]:

$$\mathbf{u(k) = \sin(\pi k/25),\ 0 < k < 251;\quad 1.0\ \ 250\ < k < 501;\quad -1.0\ \ 500\ < k < 751;\quad 0.3\sin(\pi k/25) + 0.1\sin(\pi k/32) + 0.6\sin(\pi k/10),\ k < 1001} \qquad (26)$$

Object 1: The **SISO** discrete-time nonlinear plant model is described by the following equations:

$$\begin{aligned} &\mathbf{h(k) = 0.2\,u(k)\,[1 + x_1(k) - x_2^2(k)]}\\ &\mathbf{x_1(k+1) = x_2(k)}\\ &\mathbf{x_2(k+1) = -0.15\,x_1(k) + 0.8x_2(k) + h(k)}\\ &\mathbf{y(k+1) = x_1(k+1) - 0.2x_1^2(k+1) + 0.1x_1^3(k+1) + 0.3x_1^4(k+1) + 0.5} \end{aligned} \qquad (27)$$

Simulation results, obtained for **RTNN** identification model, using saturation activation function and **BP** learning algorithm with momentum term, are given in the **Fig.1** of the Appendix and shows an excellent 10 epochs convergence (1.2% of learning and 1.4% of test **MSE**).

The second example is a discrete-time model of nonlinear mechanical object with friction, taken from [11].

Object 2: The **SISO** discrete-time nonlinear mechanical object with friction is described by the following equations:

$$\begin{aligned} &\mathbf{x_1(k+1) = x_2(k)}\\ &\mathbf{x_2(k+1) = -0.025\,x_1(k) - 0.3\,x_2(k) + 0.8\,u(k) - v(k)}\\ &\mathbf{v(k) = a\,f_1(k) + (1-a)\,f_2(k)}\\ &\mathbf{f_1(k) = b_1\,f_{11}(k) + b_2\,f_{12}(k)}\\ &\mathbf{f_{11}(k) = 0.42 - 0.18 + (1 - \exp(-x_1(k)/0.1)) + 0.5\,x_1(k)}\\ &\mathbf{f_{12} = -0.4 + 0.17 + (1 - \exp(-x_1(k)/0.1)) + 0.5\,x_1(k)} \end{aligned}$$

$$\mathbf{a} = \begin{cases} +1, & |x_1(k)| > 0.0001 \\ 0, & |x_1(k)| \le 0.0001 \end{cases}$$

$$\mathbf{f_2(k)} = \begin{cases} +0.42, & u(k) \ge 0.42 \\ u(k), & -0.4 < u(k) < +0.42 \\ 0, & u(k) \le -0.4 \end{cases}$$

$$\mathbf{b_1} = \begin{cases} +0.01, & x_1(k) > 0 \\ 0, & x_1(k) \le 0 \end{cases}$$

$$\mathbf{b_2} = \begin{cases} 0, & x_1(k) > 0 \\ +0.01, & x_1(k) \le 0 \end{cases} \qquad (28)$$

Simulation results, obtained for this plant, identified by two **RTNN** models, coordinated by a **FRBS**, are given in **Fig.2,3**. of the Appendix. The learning signal is a sum of sinusoids with different frequencies. The first and the third graphics of **Fig.2**. show the training results of both **RTNN** models (for positive and negative output signals, separately). The second graphics shows the **MSE** of training for both **RTNN** models. The results shows an excellent 20 epochs convergence with 3% **MSE** for both models. The first graphics of **Fig.3**. show the training results of the combined neuro-fuzzy model. For sake of comparison, the third graphics shows again the results given in **Fig.2**.c. for the second **RTNN** (negative output signal). The second graphics of **Fig.3**. shows the **MSE** of the combined and the second models. The results shows an excellent 20 epochs convergence with the same **MSE.**

6. CONCLUSIONS

An unified state space representation of both continuous and discrete-time mathematical models of **RTNN**, is given in two layer Jordan **RNN**. The stability, observability, controllability, learnability conditions, are also given. The paper suggests to improve the **RTNN** topology introducing a restrictive vector function in the **RTNN** feedback to preserve the **NN** stability, and saturation instead of a sigmoid function to simplify its realisation. A new improved **BP RTNN** learning algorithm, is developed. For a complex non-linear plants identification, a neuro-fuzzy approach, is used. The proposed **RTNN** and neuro-fuzzy models has been experimented by two appropriate examples.

7. REFERENCES

1. K.S.Narendra, and K. Parthasarathy, Identification and Control of Dynamic Systems using Neural Networks, *IEEE Transactions on NNs, 1(1), 1990, 4-27.*
2. K.S.Narendra, and K.Parthasarathy, Gradient Methods for the optimisation of Dynamical Systems Containing Neural Networks, *IEEE Transactions on NNs, 2(2), 1991, 252-262.*
3. P.P.C.Yip, and Y.H. Pao,. A Recurrent Neural Net Approach to One-Step Ahead Control Problems. *IEEE Transactions on SMC, 24(4), 1994, 678-683.*
4. J.T.Connor, R. Martin, and L. Atlas, Recurrent NN and Robust Time Series Prediction. *IEEE Transactions on NNs, 5(2), 1994, 240-253.*

5. P.S.Sastry, G. Santharam, and K.P. Unnikrishnan, Memory Networks for Identification and Control of Dynamical Systems. *IEEE Transactions on NNs, 5(2), 1994, 306-320.*
6. D.T.Pham,. and S. Yildirim, Robot Control using Jordan Neural Networks, *Proc. of the Internat. Conf. on Recent Advances in Mechatronics, Istanbul, Turkey, Aug. 14-16, 1995*, (O. Kaynak, M. Ozkan, N. Bekiroglu, I. Tunay, Eds., Bogaziçi University Printhouse, Istanbul, Turkey, 1995), *Vol. II, 888-893.*
7. A.C.Tsoi, and A.D. Back, Locally Recurrent Globally Feedforward Networks: A Critical Review of Architectures. *IEEE Transactions on NNs, 5(2), 1994, 229-239.*
8. I.Baruch, I. Stoyanov, and E. Gortcheva, Neural Network Models of Dynamic Processes: Stability and Learning, *Proc. of the 3-rd Int. Symp. MMAR'96, Sept. 10-13, 1996, Miedzyzdroje, Poland*, (S. Banka, S. Domek, Z. Emirsajlov, Eds., TU of Szczecin, Poland, 1996), *Vol. 3, pp. 1169-1174.*
9. I.Baruch, and E.Gortcheva, Fuzzy-Neural Model for Nonlinear Systems Identification, *Proc. of the 5-th IFAC Workshop AARTC'98, April 15-17, 1998, Cancun, Mexico*, (D.G.Nocetti, J.S. Gonzalez, P.A. Contla, P.J.Fleming, Eds., AMCA, Mexican Society of Automatic Control,1998), *283-288.*
10. Seon - Woo Lee, and Jong-Hwan Kim, Robust Adaptive Stick-Slip Friction Compensation, *IEEE Transactions on IE, 42(5), 1995, 474-479.*

APPENDIX

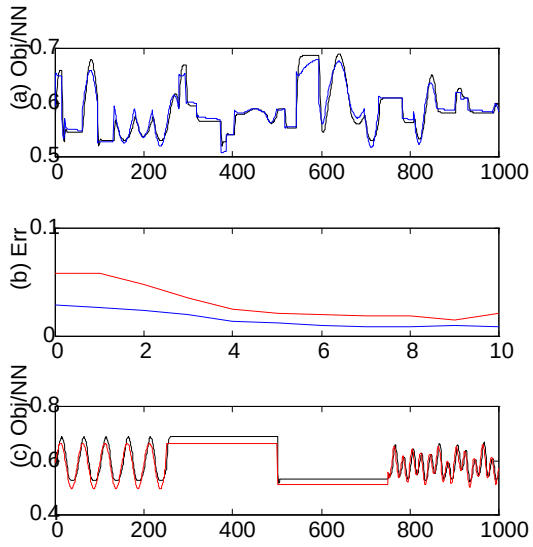

Fig.1. One **RTNN, η = α = 0.5**, saturation activation functn.
(a) *Output of **RTNN (**dashed line**)** and **Object** (solid line) during last epoch of training (1000 iterations).*
(b) ***RTNN MS*** *error of testing (dashed line) and training (solid line) for all epochs (training time - 20 epochs; MSE - 3%).*
(c) *Output of **RTNN (**dashed line**)** and **Object** (solid line) during last epoch of testing.*

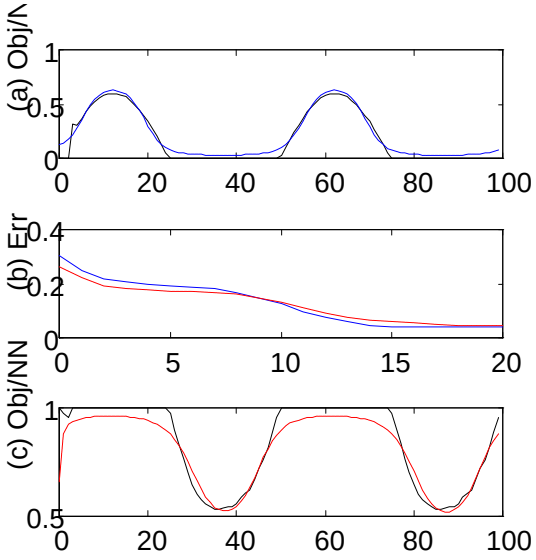

Fig.2. Two **RTNN** model approach, η = 0.2, α = 0.9, sigmoid activation function
(a) *Output of the first (positive) **RTNN (**dashed line**)** and **Object** (solid line) during last epoch of training (100 iterations).*
(b) ***RTNN MS*** *error of training the second **RTNN** (dashed line) and training the first **RTNN** (solid line) for all epochs. Time of training - 20 epochs. MSE for last epoch of training - 3%.*
(c) *Output of the second (negative) **RTNN (**dashed line**)** and **Object** (solid line) during last epoch of training (100 itrs.).*

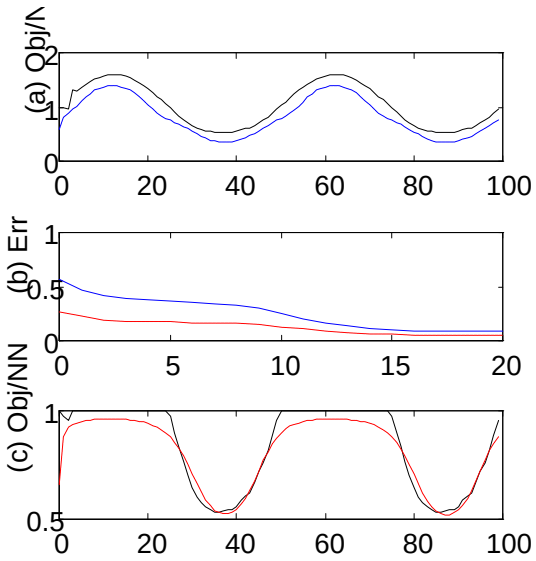

Fig.3. Two **RTNN** model approach, **η = 0.2, α = 0.9**, sigmoid activation function
(a) *Output of the combined neuro-fuzzy model **(**dashed line**)** and **Object** (solid line) during last epoch of training (100 iterations).*
(b) ***RTNN MS*** *error of training the second **RTNN** (dashed line) and training the combined neuro-fuzzy model (solid line) for all epochs. Time of training - 20 epochs. **MSE** for last epoch of training - 3%.*
(c) *Output of the second (negative) **RTNN (**dashed line**)** and **Object** (solid line) during last epoch of training (100 itrs.).*